Designing a Secret Handshake: Authenticated Key Exchange as a Capability System

Dominic Tarr

June 11, 2015

Abstract

(abstract gets written last)

1 Introduction

We need secure computer systems more than ever, part of the solution will be cryptographic – but the biggest weakness is endpoint security. Sandboxing and capability systems[3] offer hope for computer systems with simpler and more flexible security systems. Knowledge of a capability provides access to a resource. In a centralized system access may be provided by a gatekeeper when a client shows they possess a capability, but in a decentralized system that capability must be implemented cryptographically. A Capability can be a key, which enables the bearer to sign a message or encrypt something to a specific party.

Although the ultimate goal is better end point security, it's wisest to strengthen the weakest link first: currently, only Xtraffic is encrypted [**?**]

Although capability systems are a througherly good idea, there are few cryptosystems which embody a capability system (one notable example is tahoeLAFS[4]). If we are to fufil the dream of a comprehensive capability security system we need many more. Secure and private communication between two parties is one of the most fundamental crypto applications, and yet we do not have a satisifyingly simple protocol for it. TLS is complicated, contains too many ciphersuites rendering the security properties difficult to understand, and is hamstrung by a centralized Certificate Authority system.

SSH is simpler in someway, but is complexified by a layered, pluggable client authentication system.

The hardest part of a secure channel design is the handshake. let us design a handshake, suitable for a capability system, that simple enough to be described in a few lines and fully implemented in a few hundred.

- Describe a 4 pass protocol which protects all connection metadata, long term keys and even the protocol version, from any unauthenticated actor. Neither an eavesdropper, man-in-the-middle, replay attacker, or cold-caller can learn the public keys (and thus the identity) of the participants.
- The proposed handshake forms a capabality system, with 3 layers of security, knowledge of the protocol version, knowledge of the server public key, and knowledge of a secret key the server is willing to communicate with.

Development of this handshake protocol proceded from a study of the properties inherent in other protocols. This protocol uses the same elements, but arranged in a more effective way.

2 Notation

$?{\to}?meansamessagefromtheclienttotheserver.$

3 Prior Art

3.1 Unauthenticated Key Exchange

One of the simplest handshake protocols is an unauthenticated DiffieHelman exchange.

$$\begin{aligned} ? \to ? &: a_p \\ ? \leftarrow ? &: b_p \\ K_{ab} &= a \cdot b \end{aligned}$$

Alice and Bob exchange ephemeral keys, and derive a shared key. Without authentication a key exchange cannot resist a man in the middle attack, but never the less, an exchange like this is in use several popular systems. In bittorrent obsfucation the lack of authentication is inconsequential because bittorrent peers do not have any long term identity. In tor obsfproxy authentication is provided in the next protocol layer. A handshake this simple is not particularily useful unless higher layer protocols can be relied upon to provide integrety.

3.2 Authenticated Key Exchange

$$\begin{aligned} ? \rightarrow ?\ &: a_p \\ ? \leftarrow B &: b_p, B_p, Sign_B(a_p||b_p) \\ A \rightarrow B &: A_p, Sign_A(a_p||b_p) \\ K_{ab} &= a \cdot b \\ A \leftarrow B &: Box_{[a \cdot b]}(okay) \end{aligned}$$

Alice sends a fresh ephemeral key to Bob, who creates one too, signs both keys and sends them back with his public key. Since Alice just created her key, she knows the message from Bob is not a replay attack, and if she knew Bob's public key before hand then a man in the middle attack is prevented. Since *Box* is Authenticated Encryption, boxing a standard message is proof of knowing the secret, equalvalent to sending a mac.

Since Alice now knows the server is authentic, she signs the ephemeral keys and sends the signature and her public key back to Bob. Finally, to prove to Alice that she is authenticated, he encrypts an "okay" message back to Alice. In some protocols this is presented as the first message in the session body, but since Alice does not know she is authenticated with Bob until she receives it it should be considered a part of the handshake.

This design is simple and obvious, and forms the core of most common security protocols today, such as TLS and SSH. Although TLS is really a collection of protocols, when it is used with a self signed client and server certificates it is essentially this design.

Unfortunately, this design leaks both the client and server public keys. Evesdroppers can use this information to track the identities of the machines involved.

3.3 Encrypted Authenticated Key Exchange

CurveCP performs authentication based on nacl's[1] *box* primitive. Box is based on a curve25519 scalar multiplication, where $Box[content](a \cdot b)$ creates an encrypted message that can be opened by someone who knows either (a_{secret}, b_{public}), or (a_{public}, b_{secret}). I have represented this with a $\cdot$ to remind the reader that this operation is undirected. A box is between two keys, and not from one key to another – unlike RSA encryption.

$$\begin{aligned}
?\rightarrow ?\ &: a_p, Box_{[a\cdot B]}(okay)\\
?\leftarrow B &: Box_{[a\cdot B]}(b_p)\\
A\rightarrow B &: Box_{[a\cdot b]}(A_p || Box_{[A\cdot B]}(a_p))\\
A\leftarrow B &: Box_{[a\cdot b]}(okay)
\end{aligned}$$

Unfortunately, this construction has two problems.

1. because the first message is encrypted, a replay attacker may use it to confirm the identity of a server after it has moved to a different address, if it still uses the same key as before. Since only the owner of that key can decrypt that packet, if they respond it confirms they have the same identity. This could be mitigated if implementers had the server respond with nonsense, but that would be an easy detail to miss.

2. Using box for authentication like this allows Key Compromise Impersonation by anyone who knows B_{secret}. If Conrad gains B_{secret} he can and impersonate Alice to Bob. Condrad would connect to Bob, create an ephemeral key a, box it to Bob, and when Bob responds, create $Box_{[a\cdot b]}(A_p || Box_{[A\cdot B]}(a_p))$, except using (A_{public}, B_{secret}), as Bob would when opening the box, not (A_{secret}, B_{public}) as Alice would have when sealing it. If Conrad possesses B_{secret} he can impersonate arbitary keys to Bob.

3.4 Encrypted Deniable Authenticated Key Exchange

TextSecure uses a handshake that is also based on a DiffieHelman-like key exchange as is used is *Box*, but without the same Key Compromise Impersonation as in CurveCp. Here, we'll extend our notation, to use box with multiple pairs of keys. $Box_{[a \cdot b|x \cdot y]}(content)$ encrypts *content* so that it may be opened by some one that can calculate $a \cdot b$ *and* $x \cdot y$. This can be implemented by keying nacl's `secretbox` primitive with $hash(a \cdot b|x \cdot y)$

$$\begin{aligned} ? \rightarrow ? &: a_p \\ ? \leftarrow B &: b_p, Box_{[a \cdot b]}(B_p), Box_{[a \cdot b|a \cdot B]}(okay) \\ A \rightarrow B &: Box_{[a \cdot b]}(A_p), Box_{[a \cdot b|A \cdot b]}(okay) \\ A \leftarrow B &: Box_{[a \cdot b|a \cdot B|A \cdot b]}(a_p|b_p) \end{aligned}$$

This is a better design, but it has some features that make it less suitable for a capability system. Note that although it is specified as a 3 pass protocol the client does not know it is authorized until it receives the first encrypted message from the server, so again, it is really a 4 pass protocol.

1. The handshake is protected from eavesdroppers – but anyone who connects to the server will be sent the public key. This means we cannot use knowledge of the server's key as a capability.

2. KCI is still possible, but harder – To impersonate an arbitary key to Bob you have to know Bob's ephemeral *and* long term secret keys. This would be possible for anyone who had passive read access to Bob's memory – at first glance this may seem like a unlikely proposition, but in fact if Bob is running on a rented virtual machine that is precisely the situation he would be in.

This design would be more reasonable if it's use was restricted to user controlled devices. However, a general purpose protocol is likely to be ran on someone else's virtual machin. It would be worthwhile to save that someone from the temptation to interfere undetectably.

4 A New Design

If Bob never sends his public key to an unauthenticated client, or as plaintext, knowledge of Bob's key becomes a capability. If Alice preauthenticates Bob, then Bob can authenticate Alice using one more pass. With two initial passes to prevent replay attacks, we have a 4 pass protocol. This is no worse than the above, even though we do not authenticate anyone until the third pass.

To "preauthenticate" Bob, Alice sends a proof of both her identity, and her intention to connect to Bob. Preauthentication can be implemented with both encryption and signatures.

$$\begin{aligned}
?\rightarrow ?\ &: a_p \\
?\leftarrow ?\ &: b_p \\
A \rightarrow B &: Box_{[a\cdot b|a\cdot B]}(okay) \\
A \leftarrow B &: Box_{[a\cdot b|a\cdot B|A\cdot b]}(okay)
\end{aligned}$$

Requiring Alice to authenticate first is unusual, but I think this is a fair deal. Bob has already put themself at a disadvantage by allowing himself to be publically addressable. It's only fair that Alice authenticates first. By encrypting her authentication she need not reveal her identity to anyone but the one true Bob. Likewise, if Bob chooses not to accept the call, then Alice won't be able to deduce whether or not it was really Bob. Maybe it was but he did not wish to speak to her, maybe it was just a wrong number. This protects Bob from harassment.

Authenticated handshakes based on signatures do not have KCI problems like those based on key exchange. Key exchange is required for confidentiality and forward security, but signatures are required for simple resistance to KCI. Since we need both exchange and signing keys, an identity could be a pair of signing and exchange keys. nacl uses ed25519 keys for signatures, and curve25519 keys for exchange. However, nacl also provides a function to convert signing to exchange keys, so an identity could be represented as signing key. This signing key would be converted to an exchange key when necessary.

$$
\begin{aligned}
? \rightarrow ? &: a_p \\
? \leftarrow ? &: b_p \\
h &= hash(a \cdot b) \\
H &= A_p | Sig_A(B_p | h) \\
A \rightarrow B &: Box_{[a \cdot b | a \cdot B]}(H) \\
A \leftarrow B &: Box_{[a \cdot b | a \cdot B | A \cdot b]}(Sig_B(H | h))
\end{aligned}
$$

The design is getting much better. We resist eavesdropping, replay, man-in-the-middle, and KCI attacks. There are just a few minor niggles to tidy up.

If either of the two initial packets are tampered with, it would be undetected until the first authenticated packet is received. Also, if the signing keys are also used elsewhere, it's possible that a signature from this protocol or gets reused elsewhere, or vice versa [2]. These issues can be addressed by adding an application key (K_{app}) as a capability to speak this protocol. The ephemeral keys can be authenticated by using the K_{app} as the key to an *hmac*. By sign K_{app} in each signature, that will show the intention that the signature belongs within this protocol. It is vital that if there are any other cases where a signing key is used, then a similar level of care is taken to prevent ambigious interpretations of signatures created.

K_{app} could be the hash of the protocol version and the terms of service, or it simply be a random number.

$$
\begin{aligned}
? \rightarrow ? \ &: a_p, hmac(a_p, K_{app}) \\
? \leftarrow ? \ &: b_p, hmac(b_p, K_{app}) \\
H &= A_p | sign_A(K_{app} | B_p | hash(a \cdot b)) \\
A \rightarrow B &: box_{[a \cdot b | a \cdot B]}(H) \\
A \leftarrow B &: box_{[a \cdot b | a \cdot B | A \cdot b]}(sign_B(H))
\end{aligned}
$$

Alice's authentication, $A_p | sign_A(K_{app} | B_p | hash(a \cdot b))$, proves that she possesses A, and that this proof is for this protocol (via K_{app}) and for this handshake (via $hash(a \cdot b)$). Incase Bob does not have alice's key yet, she sends it with her public key.

For Bob to authenticate back to Alice, he can just sign the proof Alice sent him, and send it back. It is enough for Bob to prove that he received and decrypted this message, because Alice already knows who he is.

Alice and Bob can now use their shared secret, $a \cdot b | a \cdot B | A \cdot b$, with a bulk encryption protocol to secure a two-way communication channel.

Usually a secret key represents an *identity*, but one of the interesting things in cryptographic capability systems, such as tahoeLAFS is that a secret key more accurately a *write capability*. Systems built upon this secret handshake protocol may have shared secret keys that allow other actors to authenticate to a particular role semianonymously.

5 Conclusion

I have described a highly private handshake protocol that is suitable for capability systems. It can be described in just a few lines, and I have produced a reference implementation which is just a few hundred.

References

[1] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. The security impact of a new cryptographic library. Cryptology ePrint Archive, Report 2011/646, 2011.

[2] John Kelsey, Bruce Schneier, and David Wagner. Protocol interactions and the chosen protocol attack. In *In Proc. 1997 Security Protocols Workshop*, pages 91–104. Springer-Verlag, 1997.

[3] Tristan Slominski. Towards a universal implementation of unforgeable actor addresses. http://www.dalnefre.com/wp/2013/10/towards-a-universal-implementation-of-unforgeable-actor-addresses/, Oct 2013. Accessed: 2015-06-10.

[4] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe the least-authority filesystem. Cryptology ePrint Archive, Report 2012/524, 2012.